Due Thursday, 21 Oct 1999, at the beginning of class.

Warning! This homework is different from all previously assigned homeworks in this class.

1. There is only one problem.
2. The homework requires reading and using another programmer's code.
3. You are asked to add code to an existing program.

I encourage you to start early. Spend some time reading these instructions and the code (over six pages long!) to ensure you understand each function's purpose. Before programming, plan your work. As you modify a piece of the code, test it. This will localize any programming errors that occur. Good luck!

1 Reading

Read Sections 4.1, 5.1–5.3 and pp. 588–594.

2 Compilation Tip

Last week, we told you about the `-Wall` and `-pedantic` options for `g++` to print warning messages about possibly erroneous statements. Typing

```
g++ -Wall -pedantic hello.cc
```

everytime we want to compile a program can become tedious. Here is a better way.

Using a text editor, make a file called "Makefile" with only one line:

```
CXXFLAGS= -Wall -pedantic
```

To compile the `hello.cc` file, type

```
make hello
```

This will produce an executable program called "hello", not "a.out". Enjoy!

3 Problem description

Mr. Ab Surd would like to enter the wonderful world of computer graphics, but he is unfortunately trapped in a time warp, and the available computer equipment can only display text. Mr. Surd believes, however, that beautiful art can be created using text characters, if only the artist is sufficiently determined, so he sets out to write a simple "ASCII art" program. He begins by creating a top-down design and starting to implement it in C++, but before he can finish implementing it (i.e., writing all the code in the function definitions), he is called away by the demands of his position as Galactic Intertemporal Coordinator. Your job in this assignment is to help Mr. Surd out by filling in the gaps in his C++ program.

4 User interface

Conceptually, an ASCII-art picture is a two-dimensional grid, `nuColumns` columns wide and `nuRows` rows tall. (For the equipment Mr. Surd is using, reasonable values are `nuColumns` = 80 and `nuRows` = 24.) Each point in the grid is identified by its x-y coordinates (where the upper left corner has x=0 and y=0, and the lower right corner has x=`nuColumns` – 1 and y=`nuRows` – 1). Each point also has an associated character, which can be any text character, including a space. For example, in a completely blank picture, the character for each grid point is a space. Simple shapes (points, vertical and horizontal lines, rectangles, and squares) can be drawn on the grid by specifying their coordinates and dimensions and a character to use in drawing them. Rectangles and squares can be outline-only or filled (solid).

Mr. Surd's program allows its user to draw a picture on such a grid and display it on demand (i.e., the picture is *only* displayed when the user requests it). The program repeatedly prompts the user to enter one of a selection of one-letter commands — one for each allowed shape (point, vertical line, etc.), plus commands to display the drawing, clear the drawing, and quit. If the user enters the command to draw an object, the program prompts for its coordinates and dimensions and the character to use in drawing it.

5 Top-down design

Mr. Surd begins by defining, for each of the commands to draw a shape, an associated "command-processing function" whose job it will be to prompt for the desired coordinates, etc., and then call another "drawing" function to actually draw the shape. For example, when the user enters the `p` command to draw a point, the program invokes `doPoint()` (one of Mr. Surd's command-processing functions), which prompts the user for the point's coordinates and the character to use and then invokes `drawCharacter()` (one of Mr. Surd's drawing functions) to draw the character.

Mr. Surd notices that there are two operations that need to be done in most of the command-processing functions, namely obtaining an integer in a specified range from the user and obtaining a character from the user, so he writes a "helper function" for each of these operations, calling them `obtainInteger()` and `obtainCharacter()`.

Mr. Surd also decides to hide the details of how he implements the two-dimensional grid. That is, he defines three basic functions to operate on the grid — one to draw a single character at specified coordinates (`drawCharacter()`), one to print the grid (`printDrawing()`), and one to erase everything in the grid, that is, put spaces in all positions of the grid (`clearDrawing()`). All his other functions will make use of these basic functions. He does this because he has thought of a clever way of implementing the two-dimensional grid using a one-dimensional array, but he recognizes that (1) his other functions will be easier to understand if written in terms of operations on the conceptual two-dimensional grid, and (2) if he later changes his mind about the clever implementation, he will only need to change the three basic functions.

6 Your mission

Mr. Surd believes that he has declared (i.e., written prototypes for) all the functions needed for the program, and he has written code to fill in some of the corresponding function definitions. For the remaining functions, he simply copied their prototypes and inserted the comment `// STILL TO BE IMPLEMENTED`. You are to do the following.

1. Obtain Mr. Surd's (incomplete) C++ source. You can do this by logging into one of the CS

lab machines and copying file `/users/blm1320/ASCIIart/asciiArt.cc` to your directory.

2. Look for functions whose definitions consist of the comment `// STILL TO BE IMPLEMENTED`. Wherever you see this comment, replace it with code that implements the function as described by its prototype and the associated comments giving its precondition and postcondition.

   For example, suppose the code included a function with the following prototype and comments:

```
// precondition:  a and b are integers
// postcondition:  return value is a+b
int addTwoInts(int a, int b);
```

   and the following definition:

```
// precondition:  a and b are integers
// postcondition:  return value is a+b
int addTwoInts(int a, int b)
{
    // STILL TO BE IMPLEMENTED
}
```

   Then you would replace the comment in the function definition with the following line:

```
    return a + b;
```

   You should not need additional functions, but if you do, put their prototypes and any other needed declarations after the comment

```
// PLACE FOR ADDITIONAL DECLARATIONS
```

   and their definitions after the comment

```
// PLACE FOR ADDITIONAL CODE
```

Hint: Before beginning to write code, you should review other parts of Mr. Surd's code, especially the functions identified in the comments as "Helper functions". Mr. Surd's daughter Abby says she knows how to complete his program by adding only 45 additional lines of code. If you find yourself writing a lot more, you are probably doing something the hard way.

To help you: An executable for a sample solution is available in `/users/blm1320/ASCIIart/asciiArtSolution`; you can execute this program to see how your program should behave. (You can execute it by logging into one of the lab machines and typing its full name as given above. You can also copy it to your own directory.)

7 What to turn in

In preceding homework assignments, you were to turn in your program in the form of printed source code. For this assignment, you are to turn in your program in electronic form, by (electronically) mailing its source code to me. To do this, perform the following steps.

1. Log into one of the CS lab machines and change directories to the directory containing your source code. (If you have developed your program on another machine, copy the source code to your CS lab account. There are several ways to do this.) Suppose your source is in a file named `homework.cc`.

2. Mail the program source using the following command:

   ```
   mail -s "CS1320 homework" oldham@cs.trinity.edu < homework.cc
   ```

   This sends a message with a subject line of "CS1320 homework" and a body consisting of your source code.

Be sure that (1) your program source contains your name in the opening comments, and (2) your email message has a subject line of "CS1320 homework" — I will grade only programs that arrive in my mailbox with this subject line.

For this assignment, I will evaluate your program by:

- testing its correctness using test data I make up.
- reading your source code, looking for correctness and how well you fit your code into Mr. Surd's program.
- compiling your program. Compilation errors and warnings will decrease your score. Check your code using the `-Wall` option.